Investigating interoperability and performance portability of select LLNL numerical libraries

DOE Center of Excellence Performance Portability Meeting

Slaven Peles, John Loffeld, Carol S. Woodward and Ulrike Yang

Glendale, Arizona, April 20, 2016

LLNL-PRES-688866

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Outline

- Challenges
- Description of LLNL software stack
- SUNDIALS library
- Preliminary performance testing results
- Future work and conclusion

Challenges

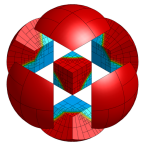
Porting existing codes to heterogeneous hardware architectures

- Implement numerical algorithms in a way that makes best use of heterogeneous hardware architecture
- Develop code that can evolve along with the new hardware -- separate platform specific from algorithmic part (RAJA, Kokkos).
- Total cost of ownership:
  - How easy is it to deploy the code in new environments?
  - How easy is it to add new features?
  - What is the maintenance cost?
- Technical debt management

Maximizing performance is but one of several challenges that need to be addressed when moving to new architectures.

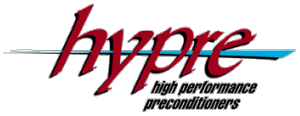
LLNL Software Stack

Libraries currently being ported to heterogeneous architectures



MFEM: A free, lightweight, scalable C++ library for finite element methods.

SUNDIALS: Suite of state-of-the-art numerical integrators and nonlinear solvers.



hypre: A library for solving large, sparse linear systems of equations on massively parallel computers

The combined use of MFEM, hypre and SUNDIALS is critical for the efficient solution of a wide variety of transient PDEs, such as non-linear elasticity and magnetohydrodynamics.

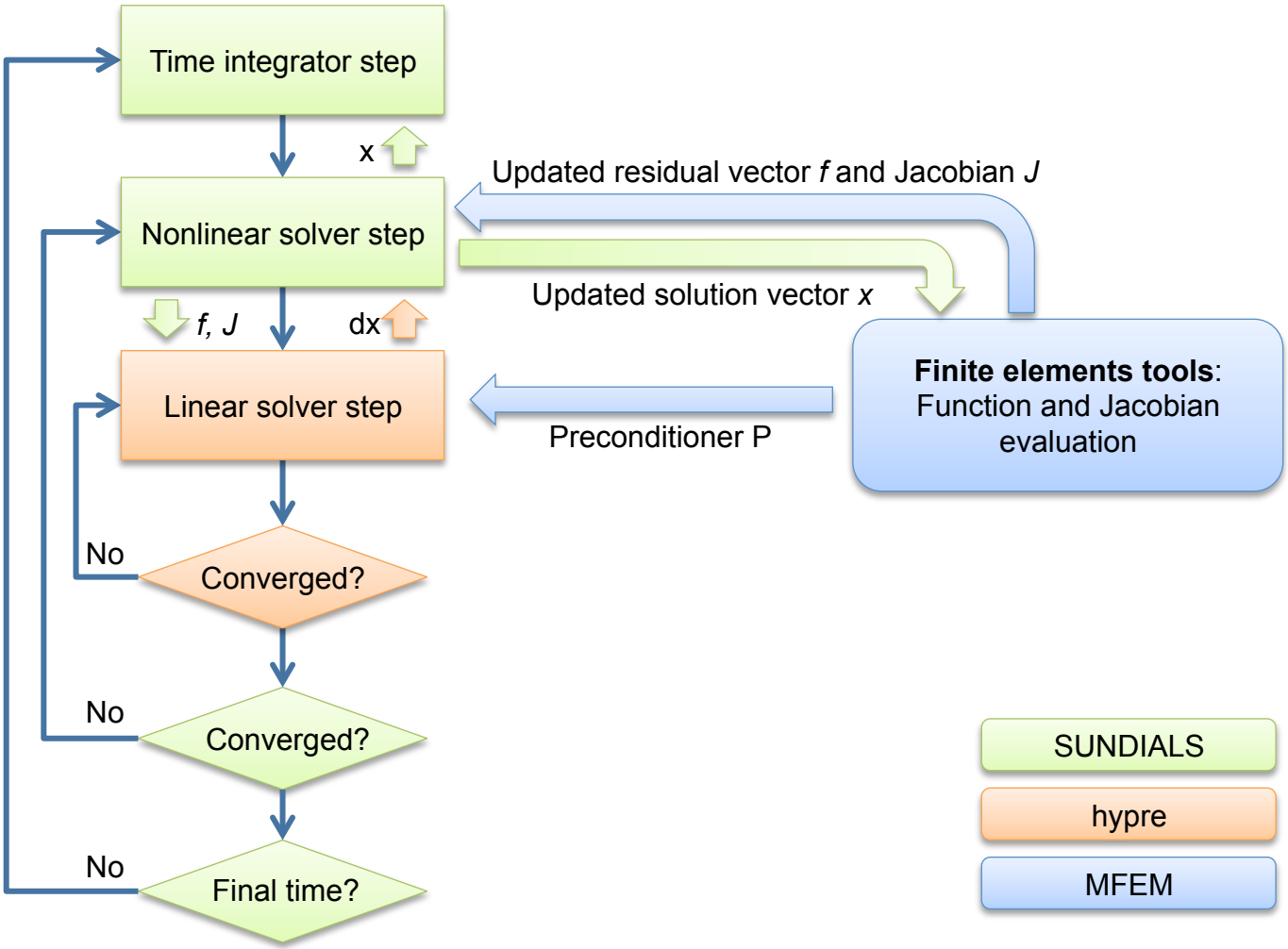
Maintaining interoperability and performance portability of the software stack is more challenging on heterogeneous architectures.

Numerical simulation and data flow

Use case: implicit integration scheme with iterative linear solver

Time integrator and nonlinear solver agnostic of vector data layout.

Numerical integrators and nonlinear solvers may invoke fairly complex step size control logic.



How to lay out the computation?

Considering constraints of heterogeneous architectures

- GPU processing power >> CPU processing power
- GPU memory << CPU memory
- Moving data between CPU and GPU is expensive
- Proposed computation layouts:

Numerical Integrator	Linear Solver	Function Evaluation
GPU	GPU	GPU
CPU	GPU	GPU
CPU	GPU	CPU
CPU	CPU	GPU
CPU/GPU	CPU/GPU	CPU/GPU

- Best layout most likely problem dependent.

SUNDIALS

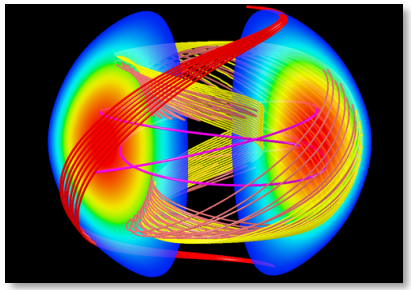
Suite of state-of-the art numerical integrators and nonlinear solvers

- Forward looking, extensible object oriented design with simple and clean linear solver and vector interfaces.
- Designed to be incorporated into existing codes.
- Modular structure allows users to supply their own data structures.
- Scales well in simulations on over 500,000 cores.
- Supplied with serial, MPI and thread-parallel (openMP and Pthreads) structures.
- CMAKE support for configuration and build.
- Freely available, released under BSD license; Over 4,500 downloads per year.
- Modules and functionality:
  - ODE integrators: (CVODE) variable order and step stiff BDF and non-stiff Adams, (ARKode) variable step implicit, explicit, and additive Runge-Kutta for IMEX approaches.
  - DAE integrator: (IDA) variable order and step stiff BDF.
  - CVODES and IDAS include forward and adjoint sensitivity capabilities.
  - KINSOL nonlinear solver: Newton-Krylov and accelerated fixed point and Picard methods.

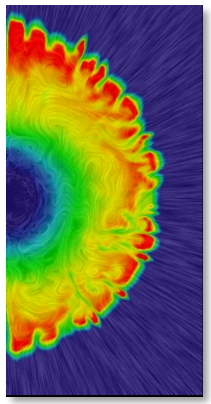
SUNDIALS

Used in industrial and academic applications worldwide

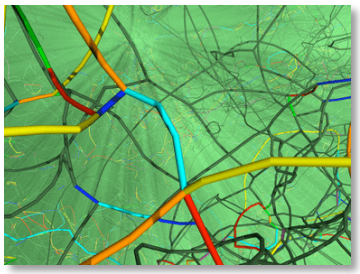
- Power grid modeling (RTE France, ISU)
- Simulation of clutches and power train parts (LuK GmbH & Co.)
- Electrical and heat generation within battery cells (CD-adapco)
- 3D parallel fusion (SMU, U. York, LLNL)
- Implicit hydrodynamics in core collapse supernova (Stony Brook)
- Dislocation dynamics (LLNL)
- Sensitivity analysis of chemically reacting flows (Sandia)
- Large-scale subsurface flows (CO Mines, LLNL)
- Optimization in simulation of energy-producing algae (NREL)
- Micromagnetic simulations (U. Southampton)



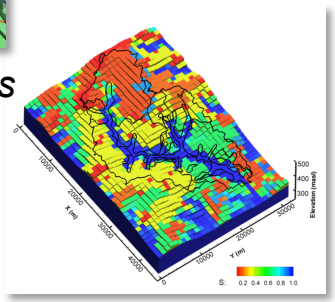
Magnetic reconnection



Core collapse supernova



Dislocation dynamics



Subsurface flow

Interfacing SUNDIALS with other software

Vector interface

- Specifies:
  - 3 constructors/destructors
  - 3 utility functions.
  - 9 streaming operators.
  - 10 reduction operators.
- Entire interaction with application data is carried out through these 19 operators.
- All are level-1 BLAS operators.
- Individual modules require only a subset of these operators.

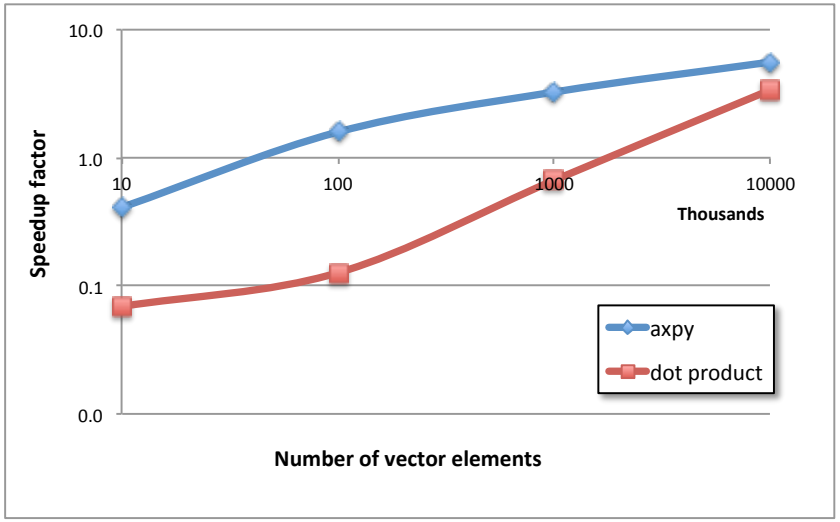
Linear solver interface

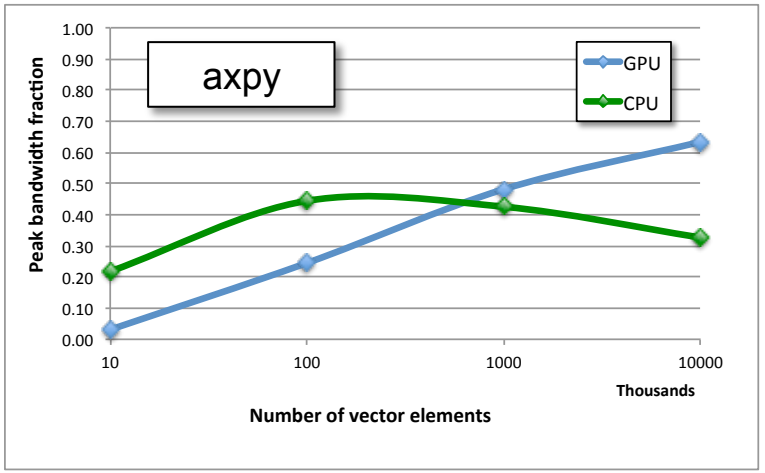
- Specifies following five functions: init, setup, solve, perf and free.
- SUNDIALS only requests linear solves at specific points. It is independent of linear solve strategy.
- Implementation of hypre linear solver interface is in progress.

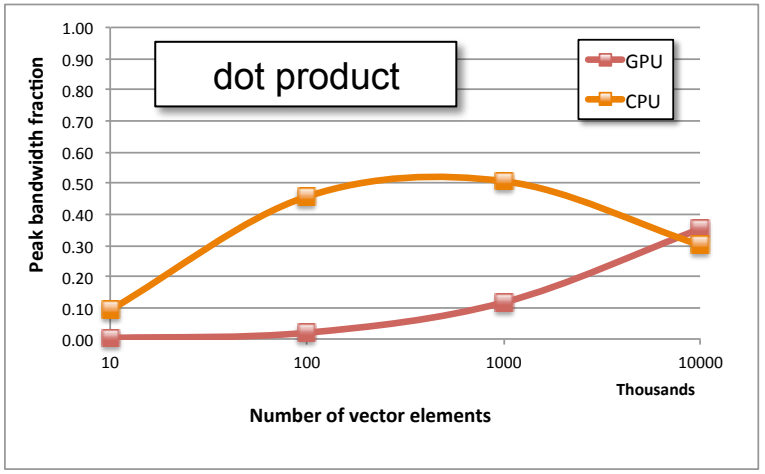
Object oriented design and well defined interfaces simplify porting SUNDIALS to new platforms.

SUNDIALS vector kernels performance testing

Standalone kernel performance





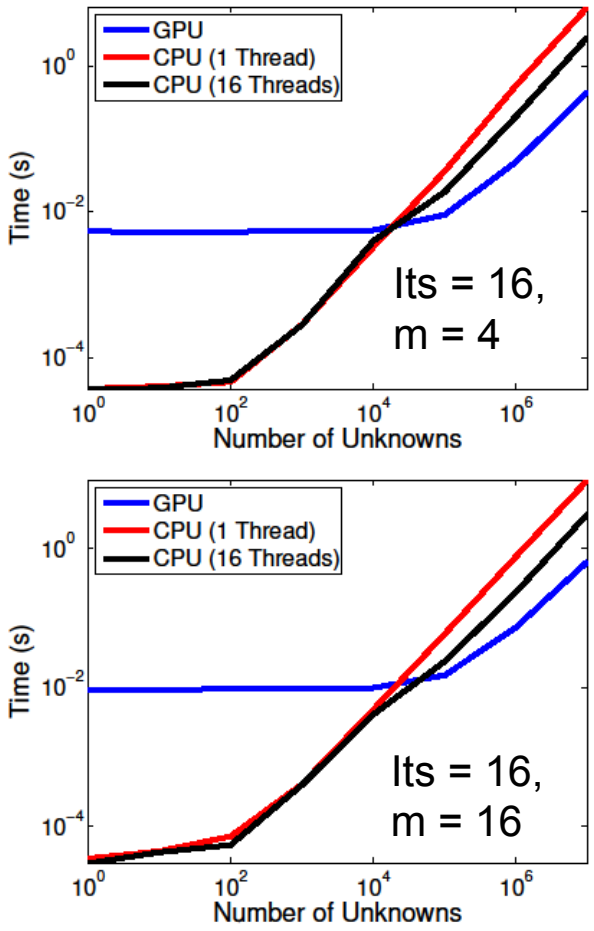


- Compared CUDA implementation with corresponding threaded MKL calls.
- Peak bandwidth of GPU is ~3x larger than CPU bandwidth on the test hardware.
- axpy shows speedup on GPU for $N > 10^4$, dot product for $N > 10^6$.

Anderson Acceleration Solver Performance

A simple nonlinear solver method implemented in SUNDIALS

Run times for CPU and GPU (function cost not timed)

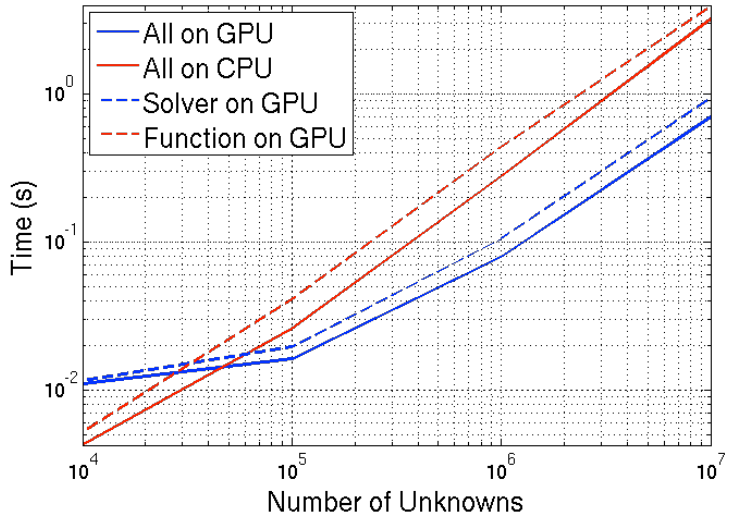


- For vectors less than 10,000, CPU versions take less time than GPU version.
- CPU version costs remain approximately constant until vector lengths reach 100.
- GPU version cost is constant until vector is 10,000 – length at which the work per vector dominates overhead per vector operation.
- Times approach linear with vector length
- When both CPU and GPU versions are in linear regime, we expect ratio between timings to be approximately ratio of bandwidth.
- Threading reduces runtime on CPU.
- GPU gives more benefit on large problem.

Anderson Acceleration Solver Performance

CPU-GPU computation layout

- Applied 16 iterations with simple function, f(x) = x; touches 2 vectors
- CPU runs used all 16 cores and 16 threads
- Timed 4 combinations:
  - Both function evaluation (FE) and vector operations on same side of bus
  - And on opposite side: CPU (GPU) = vectors on CPU, FE on GPU
- Fastest times occur when vectors are on GPU
- FE on opposite side of bus causes data transfer every iteration
- For this "light weight" function, not worth computing it on GPU if vector operations are not also on GPU (in fact, this gives worst performance)

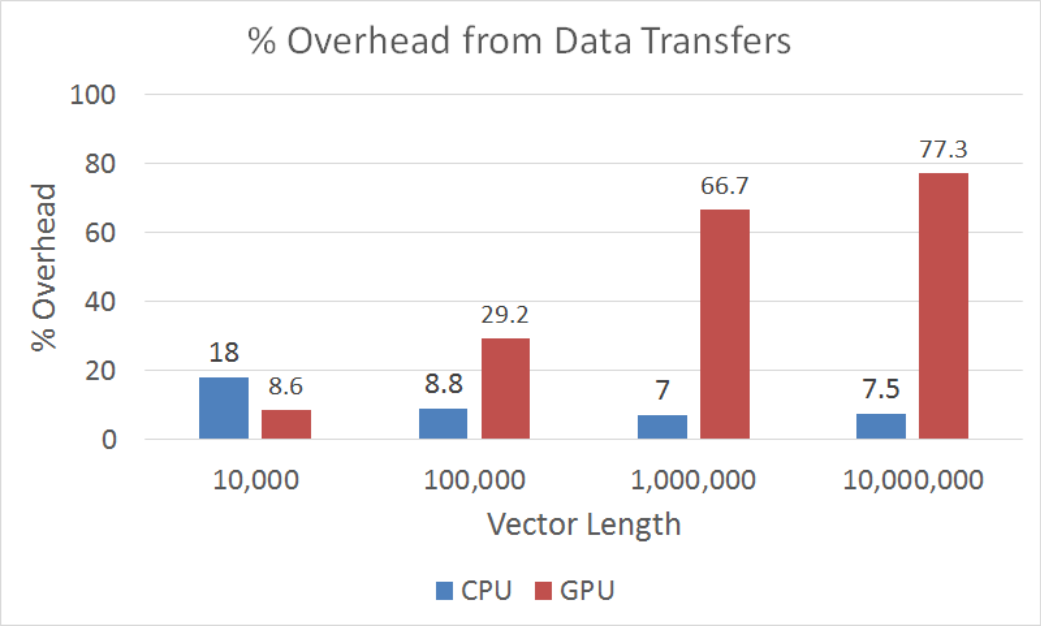


Anderson Acceleration Solver Performance

Data transfer overhead

- 16 its with m = 16
- For small vectors cost to transfer is much higher % of AA on CPU than AA on GPU
- For large vectors, cost to transfer is far larger % of AA on GPU
- AA is fast on GPU so bandwidth cost of data transfer significantly increases compared to cost of vector ops on GPU

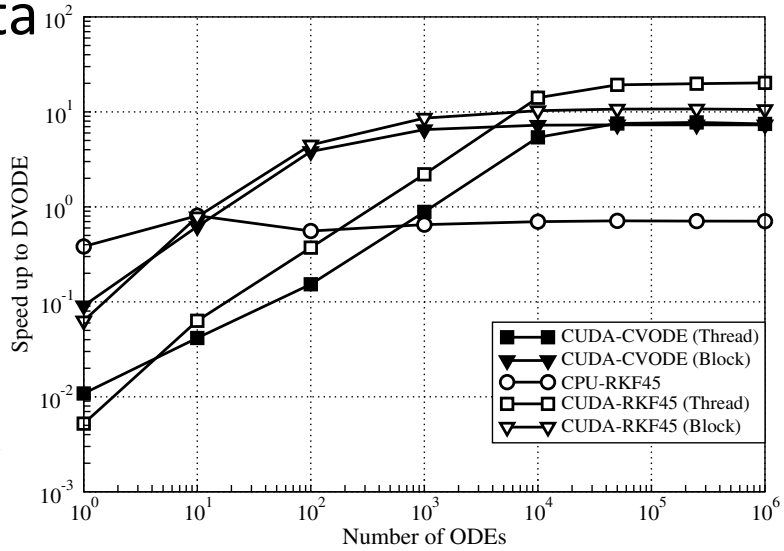
Cost to transfer one vector across PCI bus as a % cost of AA operations (mostly vector ops) for various vector sizes



Other attempts to port SUNDIALS to GPU

Stone & Davis 2013

- Test case: Multi-run simulation of a combustion kinetic problem cast in terms of stiff ODEs. Model and solver both run on GPU.
- Single ODE system small (19 equations), number of runs large.
- Used BDF (CVODE) and Runge-Kutta integrators.
- Parallelization strategies:
  - One ODE system per thread.
  - One ODE system per block.
- Reported speedup for CVODE is 7x in per block scheme, compared to serial case.



Stone, Christopher P., and Roger L. Davis. "Techniques for solving stiff chemical kinetics on graphical processing units." *Journal of Propulsion and Power* 29.4 (2013): 764-773.

Next Steps

- Prototype and implement basic data structures for the software stack (vector, matrix)
  - Implemented in hypre.
  - SUNDIALS and MFEM use wrappers around hypre objects.
- Separate partitioning from numerical algorithms
  - Parallelize using OpenMP4 pragmas
  - Use RAJA underneath
  - Use Kokkos underneath
- Maintain interoperability of the stack
- Port the stack to heterogeneous platform.
- Carry out performance analysis of the stack in different configurations and on different hardware.

Conclusions

- Heterogeneous hardware architectures are promising, but there are still many challenges to overcome. We now operate in an environment with many more degrees of freedom.
- LLNL Software Stack libraries needs to be ported as a whole. Work on individual libraries cannot be done in isolation.
- Conventional wisdom may not apply – we may need to redefine computational strategies, not simply port the same algorithms to new architectures.
- Lots of heavy lifting ahead of us: Need to create test cases, benchmark performance, analyze and improve algorithms.